

UNITED STATES PATENT APPLICATION

for

**Method and Apparatus for Fast Lookup of Related Classification Entities in a  
Tree-Ordered Classification Hierarchy**

Inventors:

Michael J. Quinn  
Mary L. Laier

prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP  
12400 Wilshire Boulevard  
Los Angeles, CA 90025-1026  
(408) 720-8300

File No.: 004683.P009

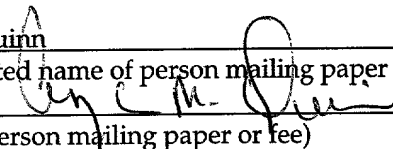
**EXPRESS MAIL CERTIFICATE OF MAILING**

"Express Mail" mailing label number: EL 617 207 615 US

Date of Deposit: December 31, 2001

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Assistant Commissioner for Patents, Washington, D. C. 20231

Angela M. Quinn  
(Typed or printed name of person mailing paper or fee)

  
(Signature of person mailing paper or fee)

12-31-01  
(Date signed)

# **Method and Apparatus for Fast Lookup of Related Classification Entities in a Tree-Ordered Classification Hierarchy**

## **FIELD OF THE INVENTION**

[0001] The present invention relates to the field of the use of a classification tree to classify traffic as part of performing network monitoring and management; more particularly, the present invention relates to the use of caching in hierarchical classification tree processing.

## **BACKGROUND OF THE INVENTION**

[0002] A traffic class is a logical grouping of traffic flows that share the same characteristics - a specific application, protocol, address or set of addresses. A flow is a single instance of a connection, session or packet-exchange activity by an identified traffic class. For example, all packets in a TCP connection belong to the same flow, as do all packets in a UDP session.

[0003] Traffic classes have the property, or class attribute, of being directional, i.e. all traffic flowing inbound will belong to different traffic classes and be monitored and managed separately from traffic flowing outbound. The directional property enables asymmetric classification and control of traffic, i.e., inbound and outbound flows belong to different classes that may be managed independent of one another.

[0004] Traffic classes may be defined at any level of the TCP/IP protocol. For example, at the IP level, traffic may be defined as only those flows between a

set of inside and outside IP addresses or domain names. An example of such a low level traffic class definition would be all traffic between my network and other corporate offices throughout the Internet. At the application level, traffic classes may be defined for specific URLs within a web server. Traffic classes may be defined having "Web aware" class attributes. For example, a traffic class could be created such as all URLs matching "\*.html" for all servers, or all URLs matching "\*.gif" for server X, or for access to server Y with URL "/sales/\*" from client Z, wherein "\*" is a wildcard character, i.e., a character which matches all other character combinations. Traffic class attributes left unspecified will simply match any value for that attribute. For example, a traffic class that accesses data objects within a certain directory path of a web server is specified by a URL of the directory path to be managed, e.g. "/sales/\*".

[0005] The classification of traffic is well-known in the art. For example, U.S. Patent No. 6,285,658 describes a method for classifying traffic according to a definable set of classification attributes selectable by the manager, including selecting a subset of traffic of interest to be classified. As described therein, the ability to classify and search traffic is based upon multiple orthogonal classification attributes.

[0006] Traffic class membership may be hierarchical. Thus, a flow may be classified by a series of steps through a traffic class tree, with the last step (i.e., at the leaves on the classification tree) specifying the specific traffic class that the flow belongs to. For example, the first step in classification may be to classify a

flow as web traffic, the next may further classify this flow as belonging to server X, and the final classification may be a match for\_URI "\*.avi".

[0007] A classification tree is a data structure representing the hierarchical aspect of traffic class relationships. Each node of the classification tree represents a class, and has a traffic specification, i.e., a set of attributes or characteristics describing the traffic, and a mask associated with it. Leaf nodes of the classification tree may contain policies (information about how to control the class of traffic). The classification process checks at each level if the flow being classified matches the attributes of a given traffic class. If it does, processing continues down to the links associated with that node in the tree. If it does not, the matching process continues on with the next (sibling) class. The last sibling in the tree must always be a default (match-all) class to catch any flows that did not match any of the classes in the rest of the tree.

[0008] More specifically, the classification tree is a N-ary tree with its nodes ordered by specificity. For example, in classifying a particular flow in a classification tree ordered first by organizational departments, the attributes of the flow are compared with the traffic specification in each successive department node and if no match is found, then processing proceeds to the next subsequent department node. If no match is found, then the final compare is a default "match all" category. If, however, a match is found, then classification moves to the children of this department node. The child nodes may be ordered by an orthogonal paradigm such as, for example, "service type." Matching

proceeds according to the order of specificity in the child nodes. Processing proceeds in this manner, traversing downward and from left to right in the classification tree, searching multiple orthogonal paradigms. The nodes are often arranged in decreasing order of specificity to permit searching to find the most specific class for the traffic before more general.

[0009]       A number of problems exist with using classification trees. First, tree-based classification is an inherently slow process. As discussed above, it requires walking the classification tree one node at a time searching for matches until a class is found that matches the incoming flow or packet. In a large tree, this becomes very time consuming. For example, in the case of an Internet Service Provider (ISP) wanting to classify an incoming flow/packet by address, when they have thousands of customers, a large amount of time would be expended walking down the tree to locate a match.

[0010]       Second, another problem that may exist is that a class tree may have a number of types of classes. For example, a classification tree may include IP addresses, services, and ports. However, there is no a-priori specification as to which of the multiple types of classes to attempt to match first.

## **SUMMARY OF THE INVENTION**

[0011] A method and apparatus for performing classification in a hierarchical classification system performing caching are described. In one embodiment, the method comprises walking a classification tree in the hierarchical classification system to determine whether an incoming flow matches a class in the classification tree, and performing a lookup on a cache storing a data structure of multiple classes of one classification type to compare the incoming flow with multiple classes at the same time to determine whether the incoming flow matches one of the classes.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

[0012] The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the invention, which, however, should not be taken to limit the invention to the specific embodiments, but are for explanation and understanding only.

**Figure 1** is a flow diagram of one embodiment of a process for performing classification.

**Figure 2** illustrates one embodiment of a network environment.

**Figure 3** is a block diagram of a network device.

**Figure 4** illustrates a representative class tree.

**Figure 5** illustrates an exemplary classification tree.

**Figures 6 and 7** illustrate adding information about a flow to a cache and later using that information for a faster lookup.

**Figure 8** is a flow diagram of one embodiment of a classification process performed by a network device.

## DETAILED DESCRIPTION

[0013] A classification tree is used to classify incoming packets. In one embodiment, the method comprises walking a classification tree in the hierarchical classification system to determine whether an incoming flow matches a class in the classification tree, and performing a lookup on a cache storing a data structure of multiple classes of one classification type to compare the incoming flow with multiple classes at the same time to determine whether the incoming flow matches one of the multiple classes.

[0014] The techniques described herein solve the problems described above with respect to the time consuming prior art process of walking a classification tree by grouping similar items together in the tree and creating a data structure for quickly determining whether the current packet/flow matches any of the items in one or more portions of the tree. In one embodiment, the data structure is a hash table and lookups are performed on the hash table to determine if an incoming flow/packet matches a class in the hash table. Almost constant-time class lookups occur when the item is found in the cache, and lookups that do not hit the cache are able to skip large sections of the classification tree.

[0015] In the following description, numerous details are set forth to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and



devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

[0016] Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0017] It should be kept in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic)

quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0018] The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0019] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement

the teachings of the invention as described herein.

[0020] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

#### *Overview*

[0021] A technique for enabling and performing a fast lookup of related classification entities in a tree-ordered classification hierarchy is described. In one embodiment, the tree-ordered classification hierarchy is a hierarchical tree structure that represents the traffic classes in a parent-child relationship. A monitoring/managing network device, described in greater detail below, uses the criteria defined by the traffic class to identify traffic flows. Traffic classes can be broadly defined or very specific. By specifying attributes of a traffic flow, such as, for example, IP address, protocol, application/service, port number, and URL, the network device characterizes the traffic types and manages them based on that characterization.

[0022] In one embodiment, the network device uses a cache memory for processing parts of the classification tree to help check flows quickly and apply policy, if appropriate. That is, portions of the classification tree are cached and

the network device encounters those portions while traversing the classification tree to determine if a match exists between the traffic classes and the incoming flow. When a cacheable portion of the classification tree is encountered, the network device performs a cache lookup to determine if the incoming flow matches one of the classes of the traffic class type that is cached. In this manner, the network device is able to check the incoming flow against multiple traffic classes at the same time, instead of performing comparisons of traffic classes to the incoming flow one traffic class at a time. Thus, the use of such a cache improves classification performance.

**[0023]** In one embodiment, the cache includes commonly used IP address based classes and uses pre-allocated memory for either IP addresses (or subnets). However, alternative embodiments may include additional or different traffic types, such as services and port numbers.

**[0024]** Figure 1 is a flow diagram of one embodiment of a process for performing classification. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or dedicated machine), or a combination of both. For example, the processing logic may be part of a network device that allocates bandwidth. In such a case, the network device identifies the traffic passing through it and matches the characteristics of the flow with traffic classes that are represented in a hierarchical tree. The tree is organized to show parent-child relationships. In other words, a class may have a subordinate child class

that has more specific criteria than its parent class. The tree lists classes in the order that the network device searches to determine to which class the given traffic flow belongs and hence, how much bandwidth to allocate to the flow.

[0025] Referring to Figure 1, the network device initially reorders the classification tree to group similar classification types together (processing block 101). In doing so, the network device keeps track of the start and end of “cacheable” areas of the tree that represent portions of the tree that are processed using cache lookups instead of node-by-node processing of the classification tree. For instance, classification by port number, classification by destination address, or classification by protocol type constitute types of classifications that could be grouped together to allow for caching. Thus, this re-ordering process allows the grouping of similar classifications together.

[0026] In one embodiment, the classification tree includes portions of the tree that include multiple classification types. These are referred to herein as portions of the tree with “mixed” classification types. During re-ordering, these portions of the tree are placed in the classification tree after all the portions of the classification tree that have a single classification type (e.g., only IP address, only service types, only port numbers, etc.).

[0027] The re-ordering of the classification tree is not necessary in cases where the tree is already ordered in a way such that similar classification types are grouped together already.

[0028] Thereafter, in response to an incoming flow or packet, normal tree-

walking is performed on the classification tree in order to classify the incoming flow (processing block 102). The processing of the classification tree occurs in a manner well-known in the art until a start of cacheable portion of the tree is encountered. At this point, a cache lookup is performed.

[0029] In one embodiment, a hash table is used and stored in the cache memory for each classification type (e.g., IP address, service, port, etc.). In the case of IP address caching, as the tree is being walked, when a cacheable area of the tree is entered, a hash function is computed. This function depends on the type of the cache (e.g., a cache of IP address classes will use a hash function on the IP address, whereas a cache for IP port classes will use a possibly different hash function which uses the IP port as the input). The application of a hash function and use of hash tables are well-known in the art. When the incoming flow is processed by the network device, as each cacheable area of the tree is entered during the tree-walk, the corresponding cache is consulted. In this manner, the network device performs classification in part by performing a cache lookup using a hash of some type.

[0030] There are one of four states of a cache lookup result:

- 1) the result is no cache entry (when class type not previously encountered by network device.)
- 2) the result is in the cache (i.e. a class value is saved in the cache),
- 3) the result is not in the cache (as indicated in the cache), or
- 4) the result is unknown (i.e., a class value has not yet been saved

in the cache.

[0031] In the first case, the result is added to the hash table with an indication that the result does not have an entry in the hash table (i.e., the cache). In other words, the result is marked as “no cache entry”. Adding an element to a hash table is well-known in the art. This case results in performing a walk through on the cacheable portion of the classification tree.

[0032] In the second case, if a class in the cacheable portion of the classification tree matches the class of the incoming flow, the result is returned from the cache. In one embodiment, the result returned from the cache is class pointer, which is indicative of user programming information that has been assigned the class.

[0033] In the third case, since an entry in the hash table was added previously to indicate that the class associated with the incoming flow is not in the cache, the network device is able to skip to the end of the cacheable area of the tree for this classification type.

[0034] In the fourth case, the network device performs a walk through on the cacheable portion of the tree. If a class is hit, then the class is stored in the cache and the result is returned (e.g., a class pointer and, thus, user programming information. If a class is not hit, then the classification engine marks the cache result as “not in the cache” (flagging that there is no cached class), and continues with the rest of the classification tree, starting at the end of the cacheable area for this cache block.

[0035] Note that the use of a hash function is not required. In the alternative, other methods of accessing an array or table representing a portion of the classification tree may be used.

#### *A Network Environment*

[0036] Figure 2 illustrates one embodiment of a network environment. Referring to Figure 2, the network environment includes a provider portion 210 of the network and a customer portion 211 of the network. Provider portion 210 of the network couples one or more servers (e.g., server 250) to the network, while customer portion 211 of the network couples one or more clients (e.g., client 251) to the network. Note that the techniques of the present invention are not limited to a network that includes client and provider portions.

[0037] Network device 220 classifies traffic flows and packets and applies preassigned policies to the flows/packets based on their classification.

#### *An Exemplary Network Device*

[0038] Figure 3 is a data flow diagram of one embodiment of a network device described herein. Referring to Figure 3, a classification engine 301 classifies traffic in the traffic flow. As described herein, classification is performed using a classification tree. The traffic may be classified using a number of types of metrics.

[0039] After classification, the traffic goes through shaping block 302 that performs any necessary shaping on the traffic. For more information on shaping,



see U.S. application serial number 08/977,376, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers," filed \_\_\_\_\_, incorporated herein by reference and assigned to the corporate assignee of the present invention.

[0040] Once classified, a bandwidth allocator 303 allocates network bandwidth to the flow/packet based on a policy assigned to the class of traffic as part of the user programming information.

#### *Classification Tree Processing*

[0041] Figure 4 depicts a representative allocation of bandwidth made by a hypothetical network manager as an example. In Figure 4, the network manager has decided to divide the network resources first by allocating bandwidth between address ranges 401-403, a service type 404 and World Wide Web server traffic class (HTTP) type. Each of address ranges 401-403 may represent a subnet. Figure 4 shows the resulting classification tree, in which address range bandwidth resources 401-403 and service bandwidth resources 404 and HTTP type bandwidth resources each have their own nodes representing a specific traffic class for that classification type. Each traffic class may have a policy attribute or other user programming information associated with it. Next, the network manager has chosen to divide the bandwidth resources of address range 401 among buildings 1-3 and the HTTP bandwidth resources among a number of addresses. Each of these nodes may have a separate user programming information associated with them.

[0042] In the case of the tree given in Figure 4, no reordering of the classification tree is needed since the address range classification types that can be used to match to a class of incoming traffic and the address range classification types are not mixed types. They are a "pure" address range that can be cached. However, if a HTTP classification type 410 is added, then address range 401 has mixed classification types and cannot be cached by itself. In such a case, reordering would place address range 401 on the tree with non-mixed cacheable classification types ahead of address range 401.

[0043] In one embodiment, the classification tree is ordered so that the pure classification types appear in the classification tree before mixed classification types. For example, in one embodiment, address classification types are encountered first during classification tree processing, the pure service classification types occur second, the pure port classification types occur third, followed by the mixed classification types.

[0044] Although the present invention is described with reference to cache host addresses, the techniques described herein are applicable to other types of classification criteria, such as, for example, but not limited to, source or destination ports, protocol types, VLAN types, packet length, and/or routing information.

[0045] One embodiment of the classification tree processing described above is given in pseudo code form below.

```
1 tclass = top of classification tree.  
2 inCache = FALSE  
while (TRUE) (
```

```

3  CACHE_PROC_PTR    cacheProc;
4  TCLASS_PTR        cacheEnd

5  if (!inCache && tclassIsCacheStart (tclass, &cacheProc, &cacheEnd)) (
6      int result = cacheProc(flowToMatch, &tclass, FIND);
7      if (result == FOUND)
8          return tclass;
9      if (result == NOTFOUND)
10         tclass = cacheEnd;
11     else /* Don't know so walk tree */
12         inCache = TRUE;
13     /* Otherwise, tclass is left at the start of the cache */
14 )
15 if (flowToMatch matches tclass) (
16     if (inCache) /* Update cache with flow info */
17         cacheProc(flowToMatch, tclass, FOUND);
18     return tclass;
19 )
20 tclass = tclass->next;
21 if (inCache && tclass == cacheEnd) (
22     /* Update cache with flow info */
23     cacheProc(flowToMatch, NULL, NOTFOUND);
24     inCache = FALSE;
25 )
26 )

```

The variable tclass indicates the location in the classification tree. The variable inCache indicates whether the cache should be used or not. In the case of “don’t know”, inCache would be FALSE and thus the classification tree would be walked. The variable cacheProc is a procedure pointer returned from tclassIsCacheStart. This procedure is called to get or set the cache entry information. The variable cacheEnd points to one passed the last class in a cache. C1 add, C2 add and C3 add indicate an entry has been added to either the C1, C2 or C3 caches respectively. The operation of the above pseudo code may be explained through the use of an example given below.

[0046] Figure 5 illustrates a classification tree. Referring to Figure 5, classes are represented using TC0 through TC7, where the DEFAULT class is a

match-all class for flows/packets that do not match the more specific classes.

Note that classification trees may include more or less classes than are shown in Figure 5. Figure 5 also shows three caches, C1 through C3, where classes TC-1 and TC-2 are C1 cacheables, classes TC-4 and TC-5 are C2 cacheables, and TC-6 is C3 cacheable. Note also that the start and end of caches and that flow, F1, is classified as TC-6.

[0047] Figures 6 and 7 illustrate two cases of how information about a flow, F1, is added to the caches (in case 1 shown in Figure 6) and later used for a faster lookup (in cases 2 shown in Figure 7). Referring to Figure 6, in case 1, the network device performs a number of passes through the “while loop” to search the classes to match the class type of the first packet of flow1 (F1) with classes in the classification tree. In case 1, F1 does not match any of the C1 or C2 cacheables. Therefore, a F1 entry is added to C1 and C2 with a NOTFOUND status set in the third and sixth passes through the while loop, respectively. This is done so that C1 and C2 can be quickly identified in case 2 (Figure 7) as areas to skip in the tree. In case 1, the notation for adding a NOTFOUND entry to a cache is “F1,12NF” where F1 is the flow number, 12 is the pseudo-code line number in the pseudo-code given below, and NF indicates it was not found in the cache.

[0048] The cache and tree walking continue in case 1 until F1 matches TC-6. Thus, the class of F1 is found and F1 is added to C3 as a FOUND entry. To elaborate, in case 1, it is the first time through the tree so the tree has to be walked until a match is found. All that is known before getting to the TC-6 is

that it is not in cache C1 and C2.

[0049] Referring to Figure 7, during case 2, a second packet of F1 is classified. In the second and third passes of the while loop, because a NOTFOUND entry was added to each of the caches, the determination that this second packet is not in the C1 and C2 caches occurs much earlier (after three passes through the while loop versus six passes through the while loop). Therefore, the determination that the second packet is found in cache C3 is able to occur more quickly.

[0050] In one embodiment, each class in the classification tree has a timestamp associated with it. In other words, when a hash table is used, a timestamp is included that indicates the last time each hash was saved. This allows an individual to modify portions of the classification table and the entire process of applying the hash function to all the classes in a cacheable portion of the tree does not have to be performed again. However, because it is possible that the classification tree can change, even if a hash is found with a pointer to a class, the time stamp must still be valid for the results of the classification to indicate that a class in the classification tree matching that of the incoming flow was found. A time stamp is considered valid if it is more recent than the last time the tree was modified.

[0051] Figure 8 is a flow diagram of one embodiment of a process performed by a classification engine in a network device as part of the classification process. Referring to Figure 8, initially, a packet is received

(processing block 801), and the classification engine obtains the next class (processing block 802). The classification engine then tests whether it is a start of a cache (processing block 803). If it is not, processing logic transitions to processing block 804, where the classification engine tests whether it is the end of the caching area. If it is the end of the caching area, the classification engine clears the "in cache" flag (processing block 805) and transitions to processing block 806. If it is not the end of the cache area, the processing logic transitions directly to processing block 806.

**[0052]** If the classification engine determines it is the start of a cache in processing block 803, processing logic transitions to processing block 809 where the classification engine performs the cache look-up. If the results of the cache look-up is a determination that the class state is unknown, then processing transitions to processing block 810 where processing logic adds to the cache by marking the entry as "unknown" and transitions to processing block 812. If the result of the cache look-up is that the classification determines that the class is in the cache, then processing logic transitions to processing block 813 where the results of the cache look-up are returned and the classification lookup is done. If the result of the cache look-up is a determination that the class is not in the cache, processing logic transitions to processing block 811 where processing logic skips to the end of the cache and transitions to processing block 803. If the results of the cache look-up are that it is unknown whether the class is in the cache or the entry in the cache that matches the class has an invalid timestamp, processing

logic transitions to processing block 812 where processing logic marks the "start of cache" as FALSE (in the pseudo algorithm this parameter is referred to as inCache) and thereafter transitions to processing block 802.

[0053] At processing block 806, processing logic tests whether the class matches. If the class does not match, processing logic transitions to processing block 802. If the class does match, processing logic transitions to processing block 807, where processing logic tests whether a class is in the cache. If the class is not in the cache, then processing logic is done. However, if the process is in the cache, the cache is updated (processing block 808) and then the processing logic is done. As part of the cache updating process, the timestamp is updated as well.

[0054] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as essential to the invention.